2017년 프로그래밍언어 개론

-02-

제출일자	2017.03.14.
이 름	정 윤 수
학 번	201302482
분 반	03

def _link_last(self, element, node):

```
def _link_last(self,element,node):
    if node.next is None:
        node.next = Node(element,None)
    else:
        self._link_last(element,node.next)
```

문제 해결 방법

link_last함수는 연결 리스트 마지막 node에 새로운 node를 연결을 시키는 함수이다. 이 함수를 재귀로 구현을 하기 위해서 base사례를 먼저 생각을 하였다. 마지막에 연결 시키기 위해서는 마지막 노드를 알아야 한다. 마지막 노드의 next의 값은 None이다 그러므로 이것을 base사례로 삼고 마지막 노드의 next에 새로운 node를 생성을 해주면 연결리스트 마지막에 추가를 시켜줄수 있다.

def _get_node(self, index, x):

```
def _get_node(self,index, x):
    if index == 0:
        return x
    elif index >0:
        return self._get_node(index-1,x.next)
```

문제 해결 방법

_get_node()함수는 Node x 로부터 index만큼 떨어져 있는 Node를 구하는 함수이다. 이것을 재귀를 구현을 할려면 base사례는 index가 0일 때 x를 반환을 해주는 것이라고 생각을 하였다. index가 양수라면 index값을 1씩 감소를 시켜주면 index만큼 떨어진 node를 얻을수 있다.

def __str__(self):

```
def __str__(self):
    if self.next is None:
        return str(self.element)
    else:
        return str(self.element) +' '+ self.next.__str__()
```

문제 해결 방법

__str__()함수는 연결리스트의 안에 있는 element들을 출력을 시켜주는 함수이다. 이 함수는 연결리스트에 있는 각 node에서 element값들을 출력을 해주어야 한다. 그러므로 link_last()를 구현을 할 때처럼 마지막 노드까지 탐색을 하면서 element 값을 반환을 해주어야 한다고 생각을 하여 이런식으로 구현을 하였다.

def __len__(self):

```
    def __len__(self):
        if self.next is None:
            return 1
        else:
            return 1 + len(self.next)
```

문제 해결 방법

__len__()함수는 연결 리스트의 길이를 반환을 해주는 함수이다. 먼저 base사례로 마지막 노드에 도달을 한다면 1을 반환을 해주면 될것이라고 생각을 하였고, 재귀가 호출이 되어질 때 마다 1을 더해주면 연결리스트의 총 길이를 알 수 있다.

def _reverse(self,x,pred):

```
    def _reverse(self,x,pred):
        if x.next is None:
            self.head = x
        else:
            self._reverse(x.next,x)
        x.next = pred
```

문제 해결 방법

_reverse()는 기존의 연결리스트의 순서를 반대 방향으로 만드는 함수이다. 연결 리스트를 뒤집기 위해서는 맨 마지막 node가 head가 되고 마지막 node의 next가 마지막 node의 이전 node를 가리키면 될것이라고 생각을 하였다. 그리하여 x의 마지막이 head가 되고 x.next=pred 로 인하여 연결리스트의 순서를 뒤집을수 있다.

def _selection(self,current_node):

```
def _selection(self,current_node):
    if current_node.next is None:
        return
    else:
        self.compare(current_node,current_node,current_node)
        self._selection(current_node.next)
```

문제 해결 방법

_selection() 함수는 compare()함수와 함께 선택정렬을 구현을 하기위해서 만들어진 함수이다. _selection() 함수에서는 연결리스트 맨 앞 node부터 마지막 node까지 하나하나 씩 방문을 하는 역할을 하면서 compare함수를 호출을 하여 현재 node와 가장 작은 element의 값을 갖는 node의 값을 바꿀수 있게 도와준다.

def compare(self, current_node, min_node):

```
def compare(self, current_node, compare_node, min_node):
    if compare_node is None:
        current_node.element, min_node.element = min_node.element, current_node.element
        return
    else:
        if min_node.element > compare_node.element:
            min_node = compare_node
    self.compare(current_node,compare_node.next,min_node)
```

문제 해결 방법

compare()함수에서는 가장 작은 값는 node를 찾은 후에 그것을 current_node의 값과 서로 swap을 시켜주는 역할을 하는 함수이다. base사례로는 마지막 node까지 모두 탐색을 해야 함으로 compare_node가 None이 될 때 까지 탐색을 하고 탐색을 하면서 가장 작은 값을 가지고 있는 node인 min_node와 current_node의 바꿔준다.

결 과

```
a b c d e
List size is 5
List size is 6
z a b c d e
d
z
a b c d e
e d c b a
List size is 10
12 61 50 46 84 26 37 14 45 95
12 14 26 37 45 46 50 61 84 95
```

느낀점

파이썬을 이용을 하여 연결리스트를 구현을 함으로써 파이썬의 객체지향적인 특성을 이해할 수 있었고 클래스에서의 메소드 생성과 객체를 이용한 메소드 호출들을 연습을 해 볼수 있었다. 또한 선택 정렬을 재귀로 구현을 해 봄으로서 재귀를 좀 더 능숙하게 사용을 할 수 있을거 같다.